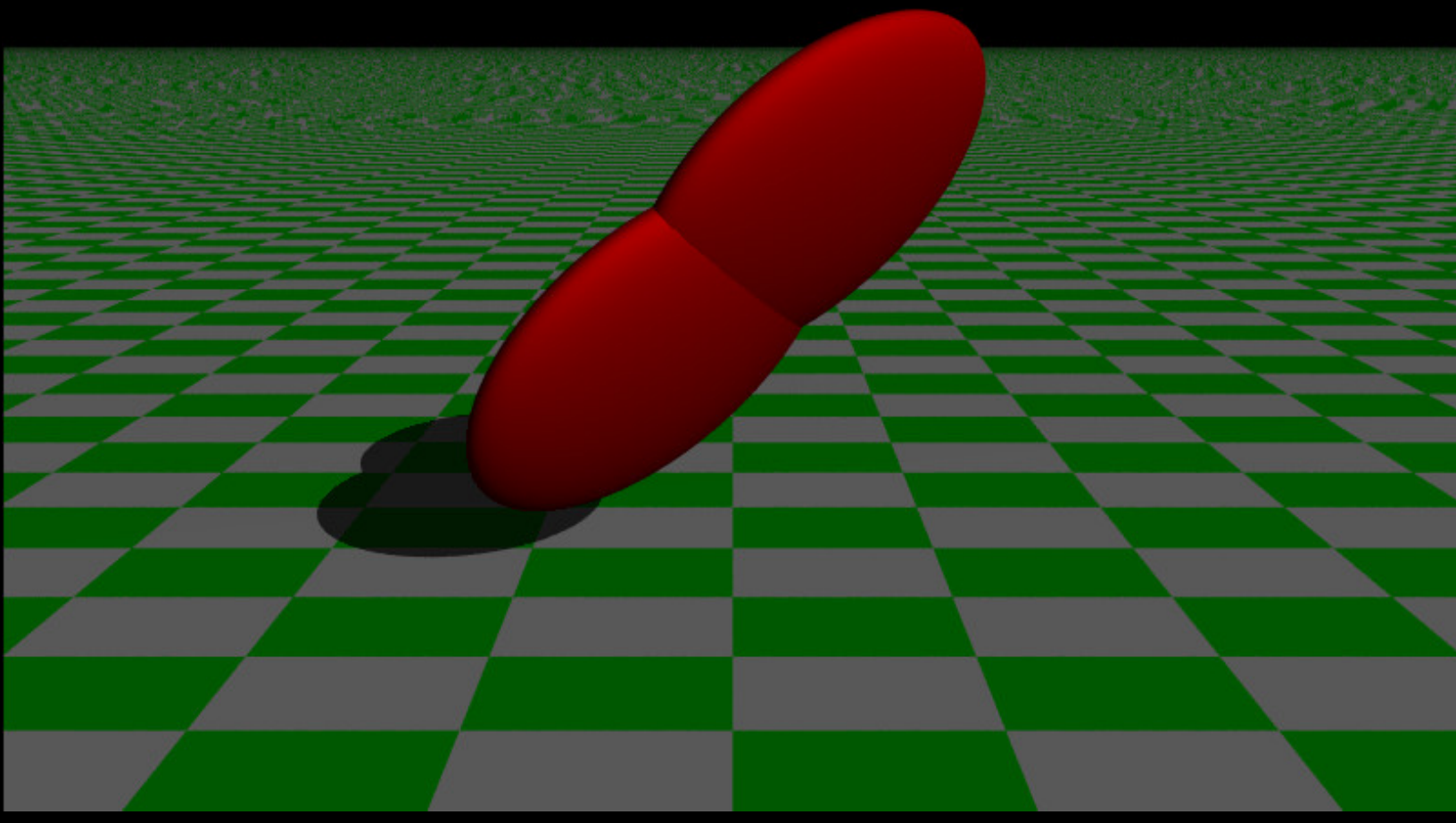
smsqzine

Issue #5 **February 2018**



Graphics with POV-RAY on the QL

SMSQzine table of contents

Published by:
Timothy Swenson
swenson_t@sbcglobal.net
swensont@lanset.com

SMSQzine is published as a service to the Sinclair QL community. Writers are invited to submit articles for publication. Readers are invited to submit article ideas.

Created using Open Source Tools:

- OpenOffice
- Scribus
- Gimp
- SMSQmulator

Editorial

When I start work on a new issue of SMSQmulator, I use the past issue as a template. Only then do I realize how long it has been since the last issue. I would offer some apology but I never did promise any set schedule, so I produce a new issue when I have enough material for the next issue.

This issue is combination of some programming work, some older software and modifying my environment to suit my needs. As most know, I do enjoy tinkering with different languages, even some of the older languages, so the articles show this preference.

The front cover is an example image created with the POV program running on SMSQmulator. It took a number of hours to generate, but it shows what is possible.

Bresenham's Circle in Fortran

After doing Bresenham's Circle algorithm in Pascal, I wanted to try it out in Fortran. Besides working with Fortran, I wanted to see what doing graphics is like using ProFortran.

```
PROGRAM CIRCLE

CALL Wopen(7,274,200,119,28)
CALL CLS(7,0)
CALL Paper(7,0)
CALL ink(7,2)
CALL bcir(100,100,20)
END

SUBROUTINE plot(xcent, ycent, x, y)
INTEGER xcent, ycent, x, y

   CALL block(7,1,1,xcent+x, ycent+y,2)
   CALL block(7,1,1,xcent-x, ycent+y,2)
   CALL block(7,1,1,xcent+x, ycent-y,2)
   CALL block(7,1,1,xcent-x, ycent-y,2)
   CALL block(7,1,1,xcent+y, ycent+x,2)
   CALL block(7,1,1,xcent-y, ycent+x,2)
   CALL block(7,1,1,xcent+y, ycent-x,2)
   CALL block(7,1,1,xcent-y, ycent-x,2)
RETURN
END

SUBROUTINE bcir (xcent, ycent, rad)
INTEGER xcent, ycent, rad, p, x, y

   x = 0
   y = rad
   p = 3 - 2 * rad

100   IF ( x .LT. y ) THEN
    CALL plot(xcent, ycent, x, y)
    IF ( p .LT. 0 ) THEN
       p = p + 4 * x + 6
    ELSE
       p = p + 4 * ( x - y ) + 10
       y = y - 1
    ENDIF
    x = x + 1
    GOTO 100
ENDIF

IF ( x .EQ. y) THEN
   CALL plot(xcent, ycent, x, y)
ENDIF
   RETURN
END
```

Using the example SQUARES_FOR, that came with ProFortran, I was able to see how to open a new window and set it up. The calls were similar to SuperBasic, in that an open call is made to open the window, and then setting the ink and paper, then doing doing a CLS.

To port the program from Pascal to Fortran, I just had to get down the Fortran specific syntax for creating subroutines and how a DO...WHILE statement is implemented (using a IF...THEN and GOTO structure). The Fortran book I have starts off with pseudo-code and then translates that into Fortran. It has a section on DO...WHILE and showed examples in Fortran.

I also had to remember that Fortran does not use mathematical symbols in the IF...THEN statements, but uses text. Where in Pascal or C there is:

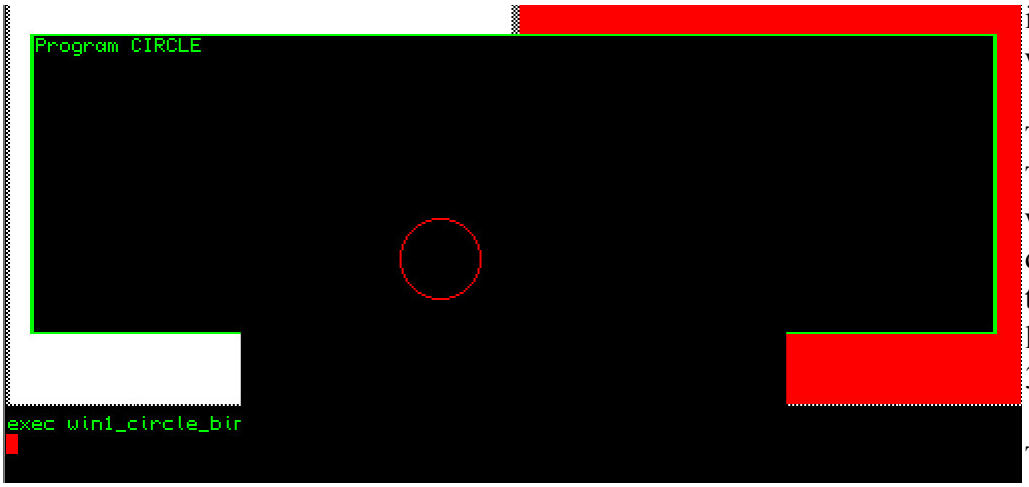
IF (x > y)

Fortran uses text like this:

IF (x .GT. y)

I was able to get the program to run. Each Fortran program compiled with ProFortran starts off with a window that asks for an input file, and output file, etc. There is no compiler option to turn this off. There is an executable program that is run against the compiled binary and this will turn it off.

There is also a console window that comes up. I looked to see if it was configurable, but no luck. I'm not sure what Fortran device it is so I can resize it and use it for my own use.



Further discussion on Fortran

While looking for Fortran 77 code to port to the QL, I ran into some issues with Fortran 77. I knew that a number of Fortran 77 compilers and books were really an update version of Fortran 77 and included some statements that were port of Fortran 90. I also learned that even in 1978, Fortran 77 was not just Fortran 77. Fortran 77 was set as an ANSI standard (X-3.9) in 1978. This was a revision from the Fortran set as a standard in 1966. There was also an addition to Fortran in late 1978 known as MIL-STD (Military Standard) 1753, which added some new functions to Fortran 77.

When looking at the book "Numerical Recipes in Fortran" which was published in 1986, it used Fortran 77 with the MIL-STD 1753 added in. Prospero ProFortran covers only Fortran 77 and does not support the functions in the MIL-STD. Since the MIL-STD was published in 1978, if I found some source code from 1978 or 1979, I could not guarantee that it would work with ProFortran, as I could have these newer functions.

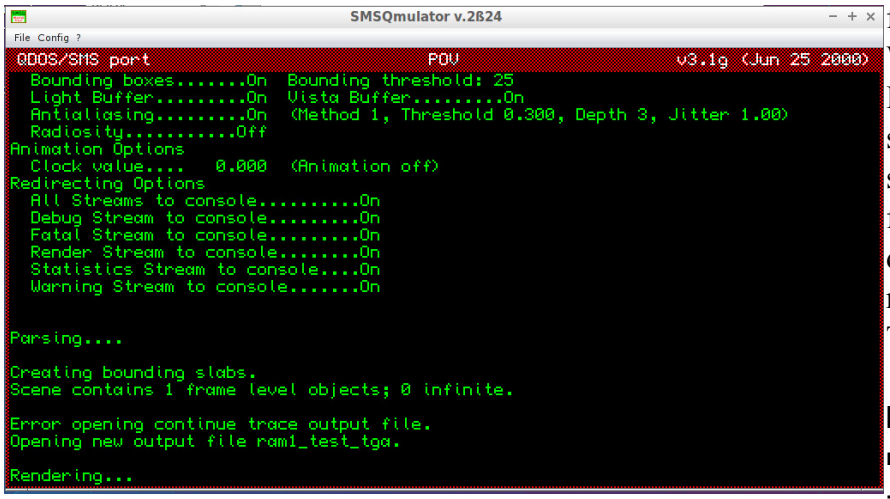
POV-Ray

POV-Ray is officially called Persistence of Vision (POV) and is a ray tracing program that has been ported to the QL by Thierry Godefroy. I've known of it since it was first ported, but I was not really interested in ray tracing. Here we are 18 years since it was ported and I thought I would give it a shot.

The version ported is 3.1g. The current version is 3.7. I was not sure if current documentation would match the older version, but I got lucky in finding a PDF of the 3.1g User Guide.

The first thing I did was to create a 10 Meg QLX.win file and put my normal boot file and boot directories on it. I made it WIN1_ and then reset SMSQmulator to boot from this QLX.win file. I then unzipped POV-RAY (pov3grun_zip) directly to WIN1_. In the distribution, the directory that POV is set to use in WIN5_POV_, but I just let the POV subdirectories go to WIN1_. The whole package is about 6 Megs once extracted.

I read through the QDOSreadme.txt file to see what I needed to do to get the software working. The document mentioned setting an environment variable for defining where the povray_ini file is located. I did that and set it in the BOOT file. I then edited the povray_ini file to change WIN5_POV to WIN1_.



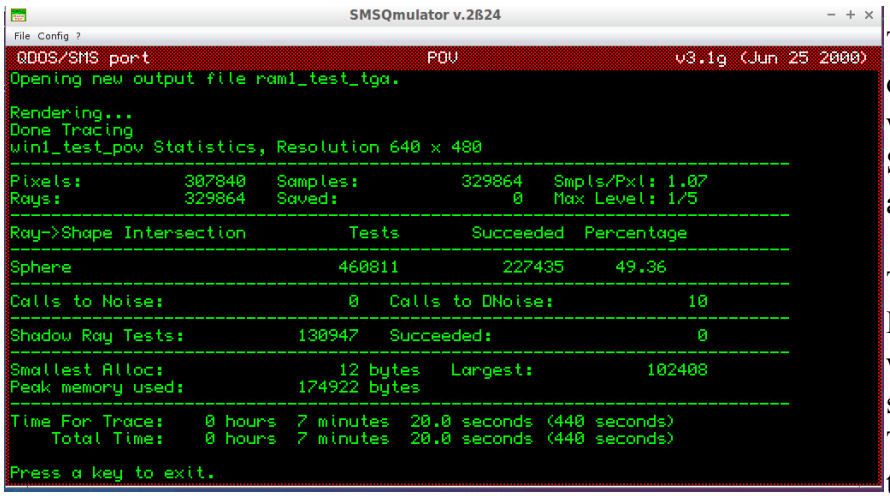
When I tried to use the example file that came with POV-ray, I got the following error "Bad Display Format". The example command line is:

```
EX win1_povray;"+iwin1_shapes_pov
+oram1_shapes_tga +d2"
```

To try out another _pov file, I used the example _pov file from the POV-Ray 3.1g User Guide. When I tried it, I also got the same error. It looked like POV was trying to execute some image viewing software. I did some digging in the User Guide and found the +d2 option was telling POV to display the image to the screen. By removing it I was skipping the issue. The command line I used was:

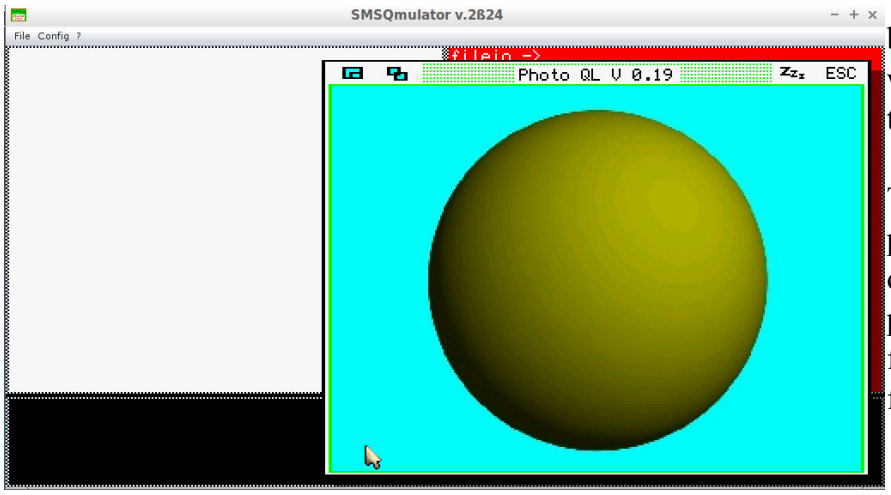
```
EX
win1_povray;"+iwin1_test_pov
+oram1_test_tga"
```

The screen shows a bunch of information as POV reads the file and then it starts to render. Once done, the time of the render is shown. In my first test case, the render took 7 minutes and 49 seconds. The output file was stored in ram1_ and was over 900K in size.



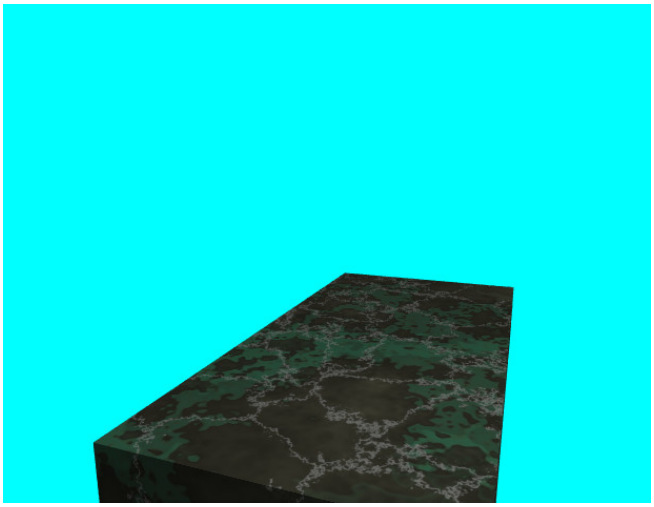
The format of the output file is TGA or TARGA. The only viewer for the TGA on the QL is PhotoQL. I loaded it and was able to view the end resultant file. Since I was using SMSQmulator, I was able to view it in high resolution.

The second test was also taken from the POV User Guide. This time the object was a box with the outside color of stone, T_Stone25 to be exact. T_Stone25 is a pre-defined stone texture that is included in stone.inc. The command line was pretty much the same, but the render time took a lot longer, a whole lot longer. With a more complex texture the render took almost 5 hours.



To keep the render time down, it is possible to change the resolution or size of the final image. Editing the povray_ini file, I changed the resolution from 600x480 to 80x60. In running the first test file again, the rendering only

took about 10 seconds. Granted the image is a whole lot smaller, but using a lower resolution is good for



debugging the POV script.
The image on the front cover was done using the test4_pov script and the render took 2 hours and 6 minutes to create. The file is 1.4 Megs. With larger renderings, instead of rendering to RAM1_, with SMSQmulator, I can render to NFA1_, which is a path to a directory at the local file system, so it can take a large image file with no issue.

If using an emulator like I was, running POV on the underlying operating system is probably going to be much faster than under an emulator, but it is still interesting to see what can be done. With some tinkering it should be possible to work on some images that would be easy to render under SMSQ/E.

```
#include "colors.inc"

background { color Cyan }

camera {
   location <0, 2, -3>
   look_at <0, 1, 2>
}

box {
   <-1, 0,-1>,     // Near lower
left corner
   < 1, 0.5, 3>    // Far upper
right corner
   texture {
      T_Stone25    // Pre-defined
from stones.inc
      scale 4      // Scale by the
same amount in all
   }               // directions
   rotate y*20     // Equivalent to
"rotate <0,20,0>"
}

light_source { <2, 4, -3> color
White}
```

Perl

Back in 1990 I had a project to read data from a 9-track tape and convert it to use on a Unix system. I could read each file from tape and store it on disk. Each file was an 80 character flat database, but there were no end of line markers. The file was just one long string. To be able to process the data, I had to break it up into 80 character lines. Shell scripts don't have a way of reading single characters. I was not well versed in C at the time. I just happened to read about a newer scripting language called Perl. When reading its manual, I found that it could read a single character at a time. I quickly downloaded the language, compiled and installed it on a system. After a little experimenting, I was able to write a script that would read in 80 characters, output 80 characters and then a EOL marker. Job done.

When the first book on Perl came out a year later, I purchased it as soon as I saw it. I started using it on another project. Soon Perl was THE System Administrator language, being used for CGI/Web work, genome work, and other projects where speed of programming is needed.

The version that I first used was Perl 3, which came out in 1989. The book covered Perl 4, which came out in 1991 and was last updated in 1993. Version 5 of Perl came out in 1995 and this has been the main

version since then. There is a big difference between Perl 4 and 5, such that most Perl 5 code will not run on Perl 4.

In 1998, Jonathan Hudson ported the last version of Perl 4 to the QL. The executable is 291K, so it needs an expanded system to run.

Perl is a scripting language that is a combination of Unix shell scripts, C, awk, and a few other languages. I viewed it as a more powerful Unix shell scripting language, that had a number of operations that shell scripts do not.

The first example of perl is the "hello world" program that everyone starts with:

```
#!/usr/bin/perl
print "Hello, World\n";
```

One use of Perl is to process a file, line by line, in just a few lines of code:

```
#!/usr/bin/perl

# name of the file to run the
report on.
$file = "test_txt";

open (INFO,$file) || die ("Failed
to open file");

foreach $line (<INFO>) {
   print $line;
}
```

One great feature of perl is using an array for a FOR statement. The normal FOR statement uses a number as in index:

FOR x = 1 to 30

Perl allows for using an array of strings. This example goes through an array and prints out each value in the array.

```
@array = ( "one", "two",
"three");

foreach $var (@array) {
   print $var
}
```

Perl is also known as a reporting language and has the concept of formatted output, where the layout of a report is designed with specified fields. This example will read though a file and sort out how many entries there are of each value.

```
#!/usr/bin/perl

$file = "list_txt";
@array = ();
open (INFO,$file) or die("Failed
to open file");

# Go through input file and
create a list
# of unique values

foreach $line (<INFO>) {
   chop($line);
   if (grep(/$line/,@array) {
     ;
   }
   else {
      join($line,@array);
   }
}

close (INFO);

## Now to go througt the file and
count each value
$count = 0;

## print header format

foreach $var (@array) {
   OPEN(INFO,$file) or die
("Failed to open file");
   foreach $line (<INFO>) {
      chop($line);
      if ($var eq $line) {
```

```
            $count = $count + 1;
      }
   ## print output format
   }
   close(INFO);
```

Perl has the concept of packages, which is sort of like a library. A package is usually a list of routines that can be used by a number of perl programs. Jonathan wrote qdos.pl that has a number of routines similar to QDOS traps. I've tested a few out but they are not working 100%. With the routine &sd_setpa that sets the screen paper, when this is set and the print statement is used, the paper on the print will be black, instead of what the paper has been set to. Here is an example:

```
#!/usr/bin/perl
require 'qdos_pl';

# Set paper color
&sd_setpa(STDOUT,-1,1);

# Clear the screen
&sd_clear(STDOUT,-1);

# Set a Green Border
&sd_bordr(STDOUT,-1,4,3);

# Set Character size
&sd_setsz(STDOUT,-1,3,1);

# Set ink color to red
&sd_setin(STDOUT,-1,2);

print "Hello\n";

&sd_setsz(STDOUT,-1,0,0);

&sd_setin(STDOUT,-1,7);

print "Hello\n";
```

Perl has some nice commands for searching using regular expressions. It also allows for search and replace in a string. When I code in SuperBasic, I use a normal editor so my code is usually all in lower case. This is the easiest way to type the code. I do want the final code to look "nicer" with SuperBasic keywords in upper case. I needed a program that would search the source code and replace all instances of keyboards with upper case versions. In looking how to do this in SuperBasic, I realized that it is not a trivial task. Using perl, it is almost a trivial task.

The script (upper.pl) reads in the SuperBasic keywords from a file and stores them in an array. Note that I used the chop() function to remove the EOL marker which is read in as part of the string. The rest of the code reads a line from the source code file, and then goes through each keyword to see if it is stored in the line. If it is found, an upper case version is put in. If the original code has uppercase, it is still replaced. There is code in the script to skip any comments, which start with either ** or ##. The code is a very brute force method of getting this done. When run under Linux, it takes hardly any time. Running on SMSQmulator, it is much slower, but still far better than doing it by hand.

The program is called like this:

```
   EXEC perl;"upper_pl < source_ssb
> source_out"
```

This reads the file source_ssb as STDIN and send STDOUT to the file source_out. The program does expect the keywords_txt file to be in the same directory. The program does not display anything when it is running. To see it working (but not saving the results), start it like this:

```
   EXEC perl;"upper_pl < source_ssb"
```

STDOUT will then be sent to the screen. A test file is included called "test_ssb".

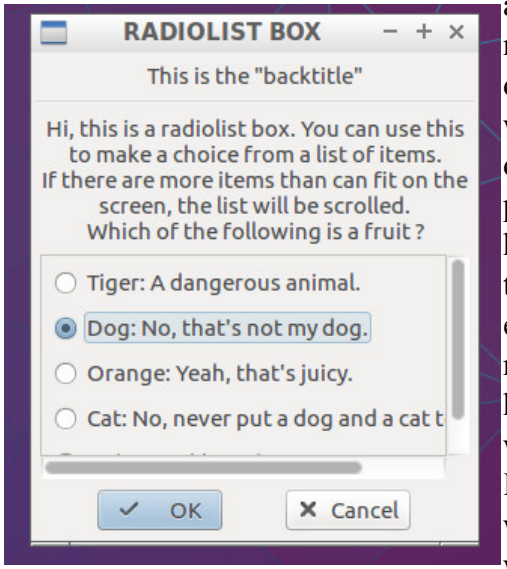
To make it faster, the keywords_txt file can be edited to remove any keywords that are not being used. It can also be added to, to include any keywords from toolkits that I did not have loaded. The file was

created by sending the output to the EXTRAS command to a file. I also had to add in the main SuperBasic command set. When adding new keywords, do not add any $ or % characters. If the keyword is "FOO$" then just add "FOO".

If you want to learn more, there are two books "Learning Perl" and "Programming Perl" by O'Reilly Books. Make sure to get the ones with the pink spine and not the blue spine. The pink book is Perl 4 and the blue book is Perl 5.

Xdialog

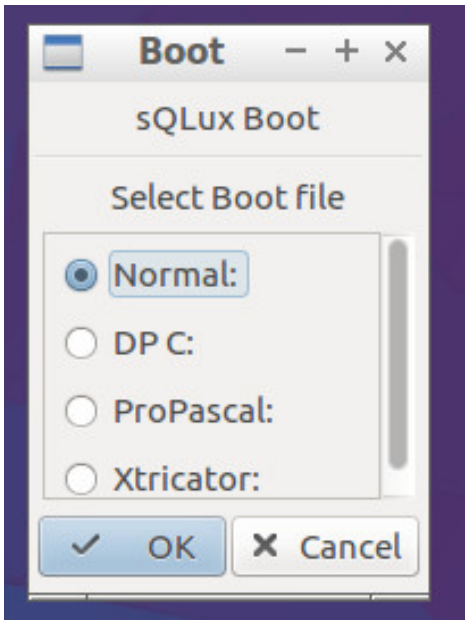
With SMSQmulator, I can add and remove QLX.win files on the fly, including the boot QLX.win file. With uQLx or sQLux, I have to edit the .uqlxrc file first, and then execute the emulator. I can't go back and add or remove devices. As I work on different projects, I have to edit the .uqlxrc file each time. I really want to have an easy way to do this. I could do what I did with SMSQmulator by having a different icon on my deskop to load a different .uqlxrc file, but after a while, my desktop would get a little crowded with all of the icons.



I did some digging on the web to try and find a X-based (GUI) menu system that is easy to implement. What I found was Xdialog, with the last version maintained by Thierry Godefroy, another QLer. Xdialog combines the bash shell with a GUI front end. It allows for a number of different GUI boxes and prompts, sort of like Qmenu.

I just wanted a simple GUI that would let me pick from a number of boot options and then fire up either uQLx or sQLux. Xdialog has simple radio buttons, where I can list a number of items, and then select one of them. Once selected, I click on OK and the script runs and I get the emulator with the boot drive that I need.

I went to the main page for Xdialog (http://xdialog.free.fr/) and downloaded the source tar file for version 2.3.1. I followed the directions for compiling the source and then installing it in the INSTALL text file. There is a directory for documentation that covers all of the types of menus that can be created. There is also a Samples directory with examples of the menus and the shell script that generates the menu.



The type of menu that I focused in on is the Radio button, where only one option of many is allowed to be chosen. From a list of 5 items, if the first item is selected, then the third option is selected, the first option will be de-selected. In the Samples directory the example script is called "radiolist".

After digging into the documentation and reviewing the radiolist script, I understood what is needed to generate the menus. The first part of the script is creating the menu and getting input from the user. The second part of the script is taking actions based on the user input.

The final script is called uqlx_xdialog.sh. Outside of the menu, it copies a .uqlxrc file of the chosen boot QXL.win file and copies it to the main .uqlxrc file so that it is used when uQLx is started. I then created a

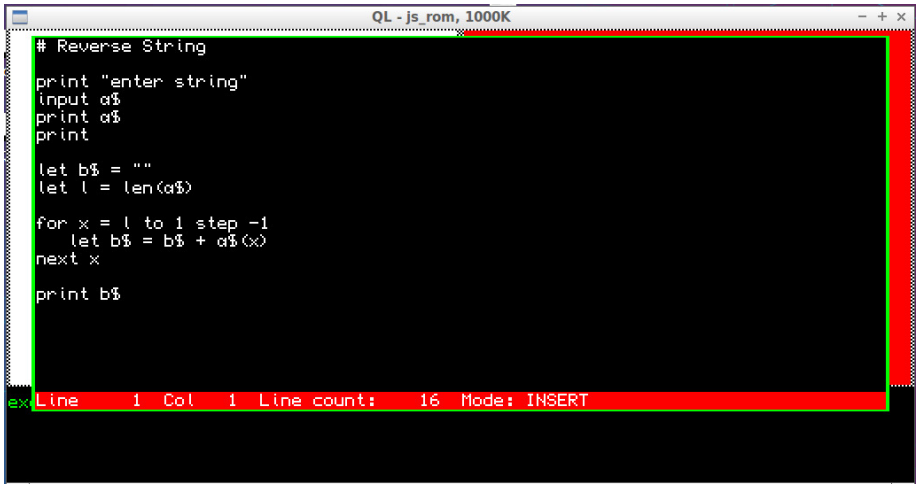
desktop shortcut that called the uqlx_xdialog.sh script. The next time I need to start uQLx, I just click on the desktop shortcut, the menu will pop up, I select the QXL.win file that I want and uQLX fires up.

ZX81 Development on the QL

I've been doing development for the ZX81 on Linux using a number of tools and a couple of ZX81 emulators. I know that Xtricator is the ZX81 emulator for the QL. It's been at least 15 years since I was using my Gold Card QL, but at that time I was not doing any ZX81 programming. A while back I

tried Xtricator and found that it will not run on SMSQmulator, so I did not go any further.

With the recent release of the new version of UQLX, which emulates QDOS and not SMSQ/E, I tried Xtricator (version 1.75) again and it worked. On UQLX, before starting Xtricator, PTR_GEN must be loaded. I load the three parts of the Pointer Environment (WMAN, PTR_GEN, HOT_REXT) and Xtricator works fine.



When running Xtricator, there are a few things to know. The keyboard is similar to the ZX81, but not quite the same. The F1 key is the way to bring up the Xtricator menu, which is very handy. Normally, SHIFT 0 (zero) is the delete or rubout key. On the QL it is CTRL <Left Arrow>. When loading a _P file from the QDOS file system, the default working directory has to be changed. The default is FLP1_ instead of WIN1_.

Memory for the ZX81 can not be initially set. Once the emulator is started, there is a menu option to reset the ZX81 and it gives a number of choices; 1K, 16K, 32K, 48K or RAMTOP Fixed.

I have tested Xtricator with a program that was compiled with Z88DK (the C compiler that can generate ZX81 code) and it ran with no issues. The emulator in an emulator does not seem to slow the ZX81 down.

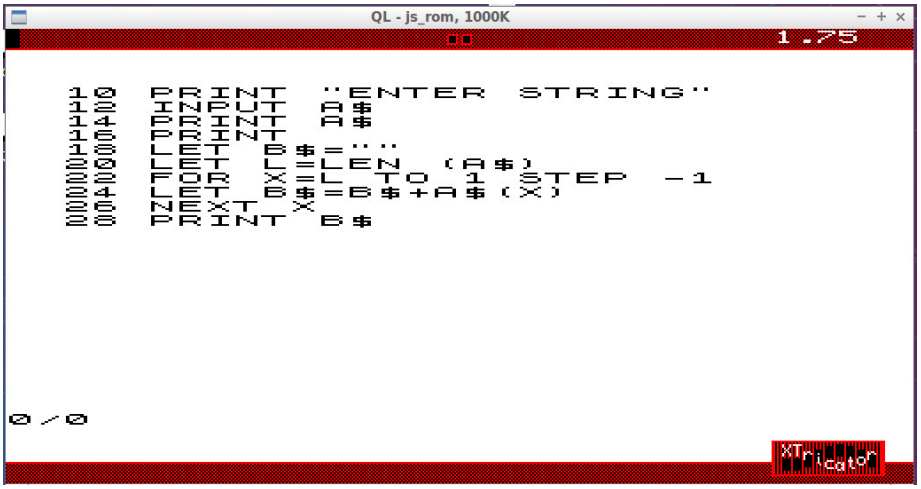
I also tested Xtricator with the a hi-res graphics program and it did not load it correctly. The pseudo-hires programs that run on a real ZX81 should run on Xtricator.

Now that I had Xtrictator running, I thought about the possibility of doing ZX81 development on the QL. The first way of doing development is in BASIC. I've been using a program called zxtext2p to convert a BASIC program written with a text editor into a binary (.P) file for the ZX81. The source code is available and it was from that source code that I compiled it to run under Linux. Using C68, I was able to compile zxtext2p to run on the QL. It is a fairly straight forward program to use.

A BASIC program is written using a text editor and can be done without using any lines numbers. Here is an example:

```
# Reverse String

print "enter string"
```

```
input a$
print a$
print

let b$ = ""
let l = len(a$)

for x = l to 1 step -1
    let b$ = b$ + a$(x)
next x

print b$
```



Anything after a # (number sign) is considered a comment and is ignored by zxtext2p. Any extra blank lines are also ignored by zxtext2p. This allows the code to have a nice look to it, but still work on ZX81 BASIC.

After typing the program into an editor like QED, save it as reverse_bas. Then execute zxtext2p like this:

```
exec zxtext2p;"-l -o reverse_p
reverse_bas"
```

A console will pop up and the phrase "Started." will appear. If there are no errors, then "Success." will be next. If there are errors, then they will be displayed. Now the _P file can be loaded into Xtricator. Once the default device had been set to WIN1_ (or what other device you use), then the program can be loaded into Xtricator like this:

```
LOAD "REVERSE"
```

zxtext2p allows for including graphics characters into BASIC. The graphics characters are first typed in ASCII and converted by the program. The all-gray characters are typed like this:

```
\##     or   \@@
```

zxtext2p does the same thing for creating inverse characters, by first creating them in ASCII.

Remember to use characters that the ZX81 supports. Bracket characters ([ ]) or braces ({ }) are not supported and should not be used.

zxtext2p was written by Chris Cowley and the short, but full, documentation can be found at the following link:

http://freestuff.grok.co.uk/zxtext2p/

Installing and Running Lynx

Lynx is a text-only web browser. The version ported to the QL is 2.8.2, which dates from 1999. It is not a great web browser, but it will demonstrate that it is possible to connect to the Internet on a QL emulator.

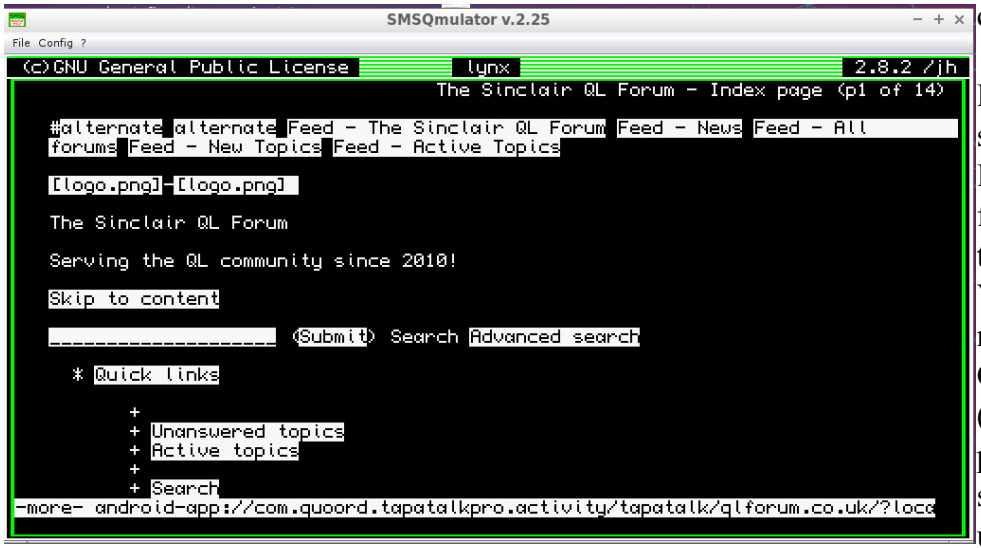
To run Lynx, I created a 2 MB lynx.win file with SMSQmulator. I then unziped the binary only version of Lynx that Dilwn Jones has created (lynxrun.zip):

```
DATA_USE WIN2_
EXEC win1_unzip;"nfa1_lynxrun.zip"
```

This put all of the files needed for Lynx on WIN2_.

Lynx requires two extensions to be loaded, environment variables (env_bin) and signals (sig30ext_rext). These come with the above

mentioned Lynx zip file. If these are not loaded in the normal boot, then they should be added at the end. Lynx also requires about 1 MB of memory on the QL. A QL with 512K, will not be able to run Lynx and there will be an "out of memory" error after executing Lynx.



There are also a number of environment variables that must be set for Lynx to run. Since I was planning on running Lynx from WIN2_, I set up the following LYNX_BOOT file that I can load from WIN2_:

```
100 DATA_USE
win2_:PROG_USE win2_
110 SETENV "TERM=qdos"
120 SETENV
"TERMINFO=win2_terminf
o"
130 SETENV
"LYNX_CON=512x256a0x0_
6_0_7"
140 SETENV
"LYNX_CFG=win2_lynx_cf
g"
150 SETENV
"LYNXRC=win2_lynxrc"
160 SETENV
"LYNX_FONT=win2_pcql_font"
```

If Lynx is installed in another path, then the boot file will need to be edited for that path.

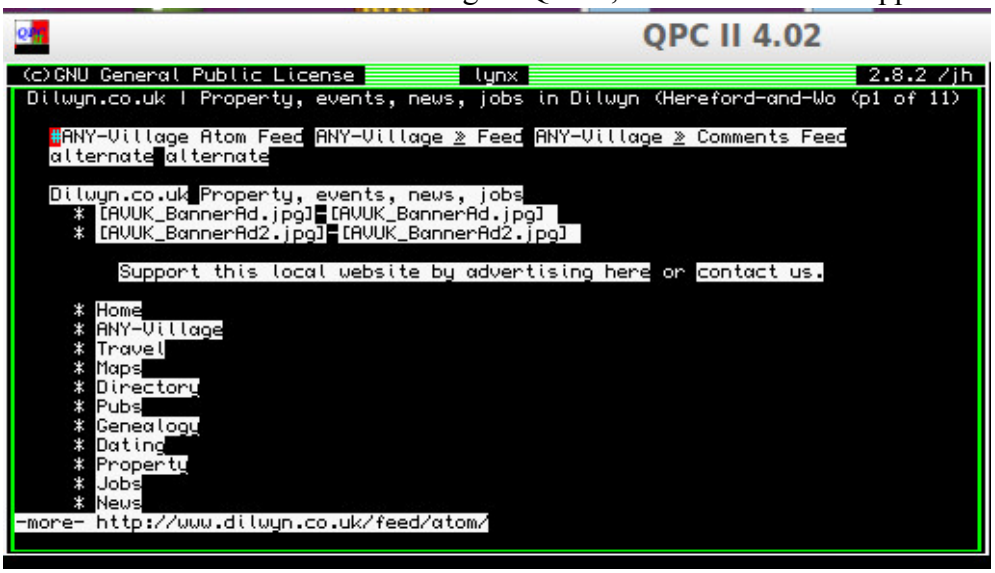
To start Lynx:

```
EXEC lynx;"www.qlforum.co.uk"
```

This will start Lynx and open the QL Forums website. There is a prompt about saving internet cookies, so just say Y for yes.

Lynx is fairly old and does not support some of the newer Internet protocols that are needed for a number of web pages. I've tried using Lynx with Google and Yahoo and both of those fail, mostly because Lynx version for QDOS does not support SSL (HTTPS:). Most of the most popular sites are now requiring secure HTTP connections by using HTTPS. Most QL related sites will probably not require HTTPS.

I've also encountered issues with Lynx where it complains about uncompressing a temporary file. I made sure that the working and data directory devices had plenty of disk space, but the issue still happened. When running on QPCII, this error did not happen.



Given how old the QL port of Lynx is, it may not be all that useful. At least it sort of works and proves that an QL emulator can get online.

For a Lynx user guide, see the following link:
http://lynx.invisible-island.net/lynx_help/Lynx_users_guide.html

I've tested Lynx with SMSQmulator, QPCII, Q-Emulator and uQLx. It worked on QPCII, SMSQmulator and Q-Emulator. It failed to run on uQLx.

uQLx and Prospero Software

Since I'm trying to go all Open Source for my computing, including QL emulation. I'm started using uQLx instead of Q-emulator, when I need to run software that will not run under SMSQ/E. The Prospero Fortran and Pascal compilers need to have a PRL ROM image in the ROM cartridge slot. uQLx supports this, but the default configuration has Toolkit II ROM image in the slot. I do not want to loose TK2, but I really need the ROM slot for Prospero.

uQLx does not support having multiple ROM's loaded (unlike Q-emulator). Luckily, there is a version of TK2 that can be load into memory and run. From Dilwyn's website I found a tk2_bin file on the Toolkit II page. I then added a part of the BOOT file to RESPR space for tk2_bin, load it into memory, and then CALL it.

The QL Report

Curry Computers of Glendale, Arizona, was a Sinclair dealer that started with the Spectrum, but switched over to the QL when it came out. When I bought my QL in 1986, I found Curry Computers as my local source for a number of QL items.

Every month, Curry Computers put out "The QL Report" which was sort of a newsletter but mostly a way of letting people know what new products were coming out. Cost was $15 for 12 issues. There was a few simple SuperBasic programs added into the newsletter to give it a little meat, but most of the writing was on new products.

Looking through my old archives (a number of ring binders), I found a number of issues of "The QL Report". I started my subscription with Volume 2, issue #10 (October 1986). Most of the issues were just full page printouts from Quill. In Volume 3 #7, Curry Computer discovers desktop publishing with Desktop Publisher, and the newsletters start looking a little bit fancier.

The last issue I have is Volume 3 #10 (Oct. 1987). I don't remember if I let my subscription lapse or if Curry Computer stopped publishing the newsletter.

Looking through the issues, I have found reference to a number of programs that were written by American programmers and published by Curry Computers. Some of them I've not heard of before. One was "The Master Teacher" by Mel MacKaron. He also authored some programming articles for "The QL Report." Steve Cermik wrote Crockett Payroll, a professional payroll system, reported to be adapted from a mainframe version.